Cycle-Detection Based Decimation Policies for Lossy Source Encoding

Masoumeh Alinia and David G. M. Mitchell
Klipsch School of Electrical and Computer Engineering, New Mexico State University, Las Cruces, NM 88003
e-mail: {aliniama,dgmm}@nmsu.edu

***Abstract*—We propose a variant of the belief propagation guided decimation (BPGD) algorithm for the lossy binary symmetric source coding problem, called DeciPolicy, which enables different decimation policies to decide when to trigger decimation, which variables to decimate, and which value to assign to decimated bits. In particular, we introduce a method that uses information about the cycles existing in the graph of a low-density generator matrix (LDGM) code to select candidate nodes for decimation. The proposed family of policies can be combined to include cycle detection-based decimation, parallel decimation of several bits, and random or hard value assignment. We demonstrate the algorithms on different constructions of LDGM codes, including an optimized irregular degree distribution and semi-regular Ising models, and show that our decimation policies lower the distortion when compared to various classical soft and hard BPGD algorithms, closing the gap to the rate-distortion limit.**

I. Introduction

In modern 5G networks and beyond, it will be critical to employ lossy source coding with limited block length to ensure ultra-reliable and low-latency communication. Lossy source coding aims to reduce the size of a given sequence in a way that allows for reconstruction up to some acceptable distortion. Linear codes can attain Shannon's rate-distortion limit for binary sources under the Hamming distance metric [1]. In [2], it was demonstrated that achieving the binary rate-distortion (RD) limit is possible through low-density parity-check (LDPC)-like codes, although their degrees increase logarithmically with the block length. Low-density generator matrix (LDGM) codes, their duals, were shown to achieve the RD limit under optimal encoding as the average node degrees increase [3]. LDGM codes share a similar representation as LDPC codes in terms of a sparse factor graph, which enables the efficient use of message-passing algorithms [4].

In practice, it is crucial to identify encoding strategies that are both simple to implement and capable of achieving the RD limit. Utilizing a conventional belief propagation (BP) algorithm falls short of delivering optimal performance for lossy source coding because of its unreliable marginal estimation in quantization. This occurs due to a large number of compressed words that result in approximately the same distortion, causing difficulty in convergence. Consequently, modifications have been made to BP algorithms, typically introducing a decimation step that periodically fixes bits and reduces the solution space in order to facilitate encoder convergence. This has been proposed in terms of either hard-decimation [5] or soft-decimation [6], as well as several variations (such as soft-hard BPGD [7]). Soft decimation achieves good distortion performance with linear complexity in code block length by substituting the hard decimation step with a soft indicator function; however, the performance of the algorithm depends heavily on the optimal parameter selection for the indicator function, which controls the distortion. In [8], a framework based on the cavity method is utilized to calculate a critical threshold for softness parameters in the soft and soft-hard BPGD algorithms. When the parameters are selected close to or equal to this threshold, the soft-hard BPGD algorithm outperforms both the hard- and soft-decimated BPGD algorithms in terms of rate-distortion performance. Asymptotically, these algorithms can display distortion performance approaching the RD limit, however they suffer in the finite length regime as a result of the cycles in the Tanner graph of the generator matrix. The corresponding message correlation during encoding results in suboptimal performance and longer convergence times. In addition, conventionally nodes are selected for decimation randomly [9], or according to their bias (likelihood) values [6], without consideration of the graph and cycle structure.

In this paper, we address this issue by introducing a novel approach called cycle detection that uses the graph structure and cycle distribution to guide the selection of variables to decimate within BPGD. It involves 'breaking' cycles in the Tanner graph by periodically identifying variable nodes that participate in the most short cycles, and fixing their variable values during decimation, thereby removing those variables from the graph structure. This can be combined with other steps to optimize the encoder convergence, such as policies for determining when to initiate decimation, how many variables to decimate, and what values to assign to the decimated variables. We demonstrate our approach on various LDGM codes, including optimized irregular LDGM codes [6] and semi-regular graphs via the Ising model [9]. Our numerical results show that certain combinations of policies involving cycle detection-based decimation, parallel and non-parallel decimation, and random or deterministic variable selection significantly outperform the state-of-the-art algorithms in terms of rate-distortion performance and complexity.

The primary objective of this paper is to introduce a framework for integrating different policies into the BPGD algorithm to improve convergence, distortion performance, and lower algorithmic complexity. Specifically, we make the following contributions:

1) We introduce a parametric approach called DeciPolicy for implementing various decimation policies within BPGD. DeciPolicy relies on three essential parameters for decimation: (i) a policy determining when to initiate decimation, (ii) a policy specifying which variables are

subject to decimation, and (iii) a policy dictating the values assigned to variables after decimation. The versatility of DeciPolicy arises from the ability to combine any policy from (i) with any policy from (ii) and (iii).
2) We present a range of decimation policies, drawing inspiration from existing methods and proposing new ones. These policies can be combined in various ways, depending on the problem at hand.
3) We implement and assess these combinations of decimation policies on various LDGM codes. We compare their performance against established methods like hard decimation and soft decimation. We observe that certain decimation policies significantly outperform existing methods in terms of solution quality and algorithmic cost.

The remainder of this paper is structured as follows: Section II provides background information on LDGM code ensembles for lossy source coding and the soft-hard BPGD algorithm [8] that inspired our DeciPolicy. In Section III, we outline the general framework of DeciPolicy and offer several examples of decimation policies. Section IV presents our experimental results and analysis of DeciPolicy, evaluating different combinations of decimation policies against soft and hard decimation algorithms. Finally, in Section V, we conclude the paper.

II. Background

A. LDGM code ensembles for lossy source coding

In this paper, we quantize a source sequence $\mathbf{s} = (s_1, s_2, \ldots, s_N)$ consisting of N independent and identically distributed Bernoulli $(p = 1/2)$ random variables to the index word $\mathbf{w} = (w_1, w_2, \ldots, w_M)$ via an LDGM code with a compression rate of $R = \frac{M}{N}$. The word $\mathbf{w}$ is used to reconstruct the source sequence as $\hat{\mathbf{s}}$, where the mapping $\mathbf{w} \to \hat{\mathbf{s}}(\mathbf{w})$ depends on the LDGM code. For the binary quantization problem, we use the Hamming metric $d(\mathbf{s}, \hat{\mathbf{s}}) = \frac{1}{N}\sum_{i=1}^{N} |\mathbf{s}_i - \hat{\mathbf{s}}_i|$ to calculate the distortion. In lossy source coding, the final goal is to minimize the average distortion $D = \mathbb{E}[d(\mathbf{s}, \hat{\mathbf{s}})]$, where $\mathbb{E}[\cdot]$ is the expectation taken over all possible source sequences $\mathbf{s}$. The rate-distortion function is in the form $R(D) = 1 - H(D)$ for $D \in [0, 0.5]$ and 0 otherwise, where H is the binary entropy function.

LDGM codes [13], as duals of LDPC codes, can be represented by generator matrix $\mathbf{G} \in \{0,1\}^{N \times M}$. We define the factor graph of this code as $\mathcal{G} = (V, C, E)$ where $V = \{1, \ldots, M\}$, $C = \{1, \ldots, N\}$, and $E = \{\ldots, (a, i), \ldots\}$ denote the code bit nodes, the generator nodes and the edges connecting them, respectively. The vector (a, i) denotes an edge between generator node a and code bit i, which occurs iff $\mathbf{G}_{a,i} = 1$. We will use indices $a, b, c \in C$ to denote generator nodes and indices $i, j, k \in V$ to denote code bits. We define the sets $C(i) = \{a \in C \mid (a, i) \in E\}$ and $V(a) = \{i \in V \mid (a, i) \in E\}$. For an LDGM code $\mathcal{C}$, defined by the generator matrix $\mathbf{G}$, and for an index word $\mathbf{w}$, the reconstructed source sequence is given by $\hat{\mathbf{s}} = \mathbf{G}\mathbf{w}$.

In this paper, we draw codes from both the Ising model construction of [9] and an optimized degree distribution [6], which is optimized in a manner similar to density evolution in the analysis of LDPC codes for channel coding. In the Ising model, each edge emanating from a regular generator node with degree k is connected uniformly at random to one of the bit nodes. The degree of bit nodes is a random variable with Binomial distribution $Bi(kN, 1/M)$ scheme, hence we refer to these codes as semi-regular. In the asymptotic regime of large N, M, the bit node degrees have i.i.d. Poisson distribution with an average degree k/R. The connections between generator nodes and code bits in an optimized LDGM code are specified by a degree distribution (λ, ρ) from the edge perspective, $\rho(x) = \sum_i \rho_i x^{i-1}$ and $\lambda(x) = \sum_i \lambda_i x^{i-1}$, where ρ_i and λ_i denote the proportion of all edges connected to generator nodes and bit nodes with degree i, respectively.

B. Cycles in a Graph

A walk of length l in $\mathcal{G}$ is a sequence of nodes $v_1, v_2, \ldots, v_{l+1}$ in $V \cup C$ such that edge (a, i) connects v_i and v_{i+1} for all $i \in \{1, 2, \ldots, l\}$. A walk is called a closed walk or a cycle if the two end nodes are identical, i.e., if $v_1 = v_{l+1}$ and if the nodes v_i, $1 \le i \le l$ are distinct. The girth g of a graph is the length of the shortest cycle in the graph. Fig. 1(a) shows an example of an LDGM graph $\mathcal{G}$, where two 4-cycles are highlighted in blue and red, respectively.

C. Soft-Hard BPGD

The algorithm used in this paper is a combination of both hard and soft decimation. Following [7], our algorithm uses soft decimation equations like [6], but after each iteration it searches for a bit node with maximum bias value. This bit node is then fixed in all future steps, and the graph is reduced with the hard decimation process. The soft-decimation BP algorithm equations are updated as follows

$$R_i^{(t+1)} = \sum_{a \in C(i)} \widehat{R}_{a\to i}^{(t)}, \quad R_{i\to a}^{(t+1)} = \sum_{b \in C(i)\setminus a} \widehat{R}_{b\to i}^{(t)} + \frac{1}{\mu} R_i^{(t)}, \tag{1}$$

$$\widehat{R}_{a\to i}^{(t+1)} = 2(-1)^{s_a+1} \tanh^{-1}\left(\beta \prod_{j \in V(a)\setminus i} B_{j\to a}^{(t)}\right), \tag{2}$$

$$B_i^{(t)} = \tanh\left(\frac{R_i^{(t)}}{2}\right), \quad B_{i\to a}^{(t)} = \tanh\left(\frac{R_{i\to a}^{(t)}}{2}\right), \tag{3}$$

where $R_{i\to a}^{(t)}, \widehat{R}_{a\to i}^{(t)}$, and $B_{i\to a}^{(t)}$ denote the message sent from code node i to check node node a, the message sent from check node a to code node i, and the bias associated with $R_{i\to a}^{(t)}$ at iteration t, respectively; $R_i^{(t)}$ and $B_i^{(t)}$ denote the likelihood ratio of code bit i and the bias associated with $R_i^{(t)}$, respectively; and $\beta = \tanh(\gamma)$ and μ are non-negative parameters. The γ parameter reflects the effort of the message-passing algorithm to find the resulting codeword $\hat{\mathbf{s}} = \mathbf{G}\mathbf{w}$ as close to $\mathbf{s}$ as possible. The larger the γ, the stronger is the effort. On the other hand, the structure of the code imposes a limit on how strong this effort can be. The term $\frac{1}{\mu} R_{i\to a}^{(t)}$ is added to the plain BP equation to make the decimation softer.

The soft indicator function

$$I_S(B_{i\to a}^{(t)}) = \frac{2}{\mu} \tanh^{-1}(B_{i\to a}^{(t)}) = \frac{1}{\mu} R_{i\to a}^{(t)}$$

approximates the hard-indicator function [7], given as

$$I_H\left(B_{i\to a}^{(t)}\right)=\begin{cases}-\infty, & B_{i\to a}^{(t)}=-1,\\ 0, & -1<B_{i\to a}^{(t)}<1,\\ +\infty, & B_{i\to a}^{(t)}=1,\end{cases}$$

where μ controls the softness of the approximation and is called the *softness parameter*. Algorithm 1 describes the procedure, where t indicates the iteration number, $\mathcal{G}^{(t)}$ is the LDGM code graph at iteration t, and w_i represents the binary value assigned to code node i. The initial information to check node node messages, $R_{i\to a}^{(0)}$, are set to ± 0.1 with $\mathbb{P}\left(R_{i\to a}^{(0)}=0.1\right)=0.5$, and reset to 0 at iteration 1.

Algorithm 1 Soft-Hard Decimation Algorithm

Require: At iteration $t=0$, initialize graph instance $\mathcal{G}^{(t=0)}$;
  Generate a Bernoulli symmetric source word s;
  while $V\neq\emptyset$ **do**
    Update $R_{i\to a}^{(t+1)}$ according to (1) for all $(a,i)\in E$;
    Update $\widehat{R}_{a\to i}^{(t)}$ according to (2) for all $(i,a)\in E$;
    Compute bias $B_{i\to a}^{(t)}$ and $B_i^{(t)}$ according to (3);
    Find $B^{(t)}=\max_i\left\{\left|B_i^{(t)}\right| \mid \text{i not fixed}\right\}$;
    if $B^{(t)}>0$ **then**
      $w_i\leftarrow '0'$;
    else
      $w_i\leftarrow '1'$;
    end if
    $\forall a\in C(i), s_a\leftarrow s_a\oplus w_i$ (update source);
    Reduce the graph $\mathcal{G}\leftarrow\mathcal{G}\backslash\{i\}$;
    $\mathcal{G}^{(t+1)}=\mathcal{G}^{(t)}\backslash\{i\}$ (remove code node i and all its edges);
  end while

In [6], there is no analytical way to tune the value of μ or β in order to give the best distortion performance. There, and in subsequent papers, e.g., [7], the best value of μ is found numerically by exhaustive and expensive code simulation. In [8], spin glass theory in the cavity method is applied to carefully tune the softness parameters in the soft-hard BPGD algorithm to ensure good RD performance.

III. Decimation Policies for BPGD

The emphasis of our framework revolves around the concept of decimation, which involves augmenting the BP algorithm by reducing the solution space so that the algorithm will converge - typically by assigning a value to a variable. Consequently, our framework addresses three pivotal questions: (i) determining the time to trigger decimation, (ii) selecting which variable(s) to decimate, and (iii) establishing the values to assign to the decimated variable(s). Each of these questions can be addressed through various criteria, and DeciPolicy outlines these criteria as decimation policies, which serve as parameters for the decimation process.

A. Motivational Example

The performance of message-passing algorithms is significantly influenced by specific graphical structures within the Tanner graph of the code, particularly short cycles and related objects such as trapping sets. Researchers have spent significant effort to design LDPC codes for channel coding problems avoiding such structures. In the following example, we show that cycles are similarly detrimental for BPGD algorithms on LDGM code graphs.

Example III.1. We consider two codes of length $N=2100$, drawn from the (3,6)-regular LDGM block code ensemble with $R=1/2$ and lifting size 350. Code A has the cycle distribution $N_4=700$, $N_6=1050$, $N_8=1400$, and $N_{10}=17500$, where N_l shows the number of cycles with length l, while code B has $N_4=2100$, $N_6=2800$, $N_8=23800$, and $N_{10}=149100$. Applying Algorithm 1, we obtain $D_A=0.379524$ and $D_B=0.394762$, respectively.

In this paper, rather than optimize the code graph to remove cycles, we focus on selecting a decimation policy that can reduce the impact of cycles in a general code graph.

B. DeciPolicy

We refer to the factor graph state at iteration t as $\mathcal{G}^t$. This state represents the combined status of all the data structures utilized by the BP-based algorithm to work with the associated factor graph. These data structures encompass various elements such as the marginal values w_i, a value w_i^* for the corresponding variable, the messages exchanged between variables (R_{ij}), and the collection of variables that have been decimated, labeled as $\mathcal{W}$. We designate the set encompassing all possible factor graph states as S.

Definition 1. *A decimation policy is a tuple $P=\langle\tau,\psi,\chi\rangle$ where:*

- $\tau: S\to\{0,1\}$ *represents the condition required to initiate the decimation process, specifically referred to as the trigger policy;*
- $\psi=\langle\phi,\theta\rangle$ *represents the policy for selecting variables to decimate, where $\phi: S\to F\subset V$ is a filtering policy responsible for choosing possible candidate variables for the purpose of decimation, and $\theta: F\times S\to V'\subset V$ represents the condition that dictates when to carry out variable decimation, known as the perform policy;*
- $\chi: v\times S\to\{0,1\}$ *is the policy used for assigning a value to a given variable $v\in V$ known as the assignment policy.*

This framework can accommodate a diverse set of decimation-driven algorithms by specifying distinct decimation policies. For example, one may consider a DeciPolicy instance, where it initiates decimation after a predefined number of iterations (τ function), randomly selects a variable for decimation from the entire pool of non-decimated variables (ψ function), and determines the value for the decimated variable by sampling it according to its marginal probability values (χ function).

The DeciPolicy framework is incorporated in BPGD as Algorithm 2. In the next subsections, we detail the three criterion τ, ψ, and χ.

Algorithm 2 Soft-Hard DeciPolicy Algorithm

Require: A factor graph $\mathcal{G} = (V, C, E)$, policy $P = \langle \tau, \psi, \chi \rangle$, source sequence $\mathbf{s}$, I_{max}

1) Initialize BP messages, $t \leftarrow 0$,
2) $\mathcal{W} \leftarrow \emptyset$, $w^* = \underline{0}$
3) **while** $\tau(\mathcal{G}^t) = 0$ and $t < I_{max}$ **do**
4) Iterate BP following equations (1)-(3)
5) $t++$
6) **end while**
7) **while** $\mathcal{W} \neq V$ and $t < I_{max}$ **do**
8) Choose node set $V' = \psi(\mathcal{G}^t)$
9) **for** $w_i \in V'$ **do**
10) $w_i^* \leftarrow \chi(w_i, \mathcal{G}^t)$
11) $\mathcal{W} \leftarrow \mathcal{W} \cup \{w_i\}$
12) Update $\mathcal{G}^t$ (remove code node i and all its edges)
13) $t++$
14) **end for**
15) **end while**
16) **if** $\mathcal{W} \neq V$ **then**
17) Assign values to remain variables w^* in $V \backslash \mathcal{W}$
18) **end if**
19) return w^*

C. The Trigger Policy τ

The trigger policy τ takes input $\mathcal{G}^t$ and outputs 0 or 1, indicating decimation should not, or should, begin, respectively. In the initial method introduced by [10], decimation is initiated when BP can no longer proceed, i.e.,

$$\tau_{\text{converge}}(\mathcal{G}^t) \overset{\text{def}}{=} \begin{cases} 1, & \text{if } \mathcal{G}^t \text{ is quiescent,} \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

This triggering mechanism involves identifying the quiescent (dormant) state of the current factor graph. We propose to use a variant of this that is based on the change in message (bias) values. Formally, the condition for triggering decimation is defined as follows:

$$\tau_{\text{bias}}(\mathcal{G}^t) \overset{\text{def}}{=} \begin{cases} 1, & \text{if } \nu < \epsilon, \\ 0, & \text{otherwise,} \end{cases} \tag{5}$$

where $\nu = \frac{1}{|E|} \sum_{(i,a) \in E} \left| \widehat{R}_{a \to i}^{(t+1)} - \widehat{R}_{a \to i}^{(t)} \right|$ is the average difference of messages in two successive iterations and $\epsilon > 0$ is some small positive number. We will use this decimation trigger in our soft-hard decimation algorithm where we check if the total number of messages does not change significantly in two successive iterations. The message values are determined by (2).

In specific scenarios marked by strict time or computational constraints, waiting for the convergence of BP may be impractical. Consequently, one might contemplate decimating before reaching convergence. The alternative strategy often depends on the iteration index. For example, in [9], decimation begins directly, e.g., $\tau(\mathcal{G}^t) = 1 \ \forall t > 0$ but, for example in [7], decimation is delayed as $\tau(\mathcal{G}^t) = 1$ where $t > I_{\text{init}}$ for some integer I_{init}.

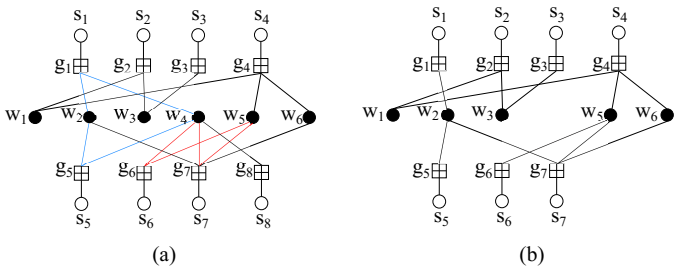

Fig. 1. Cycle-detection policy to identify a potential code bit to decimate.

D. The Perform Policy ψ (Cycle-Detection)

Now that our algorithm has the capability to determine when decimation should occur, the next question is: "Which variables should be decimated?" The $\psi = \langle \phi, \theta \rangle$ policy takes as input a graph state $\mathcal{G}^t$ and the variable node set V. The "filter" $\phi(S)$ returns the set $F \subset V$ containing variable nodes that are to be considered for decimation. Then $\theta(F, \mathcal{G}^t)$ returns the set $V' \subset V$ of variable nodes that should be decimated. In contrast to the hard BPGD approach [9], it is possible to simultaneously decimate multiple variables, e.g., $|V'| > 1$, as seen in certain variations of [7]. Moreover, the selection of variables to be decimated is not random but rather informed, determined by the ψ criterion.

In previous works (e.g., [6]- [9]), the variable selection was independent of the graph structure. In those works, the ϕ filter would output the set $F = V \backslash \mathcal{W}$, i.e., all non-decimated nodes over candidates, we refer this policy as ϕ_{all}; now, θ selects the node(s) from the filter to decimate. One approach is for θ to select z nodes randomly and uniformly from the filter output; we refer to this policy as $\theta_{\text{rand}}(z)$. Another (more complex) possibility is to search over all nodes that pass the filter and decimate the z nodes with largest bias; we refer to this policy as $\theta_{\text{maxbias}}(z)$. If there are multiple candidates with equally large bias, we select the z nodes from them randomly and uniformly. Both hard BPGD [9] and soft-hard BPGD [7] use $\theta_{\text{maxbias}}(z)$, conventionally with $z = 1$.

We now introduce a new approach (cycle-detection) that selects candidates for decimation from among those with largest bias but with the extra condition that we pick those that participate in the most short cycles in the factor graph. In this context, decimation considers cycles, making it a suitable strategy to break these cycles and improve the algorithm performance. Fig. 1 illustrates this approach. We observe from Fig. 1(a) that, of all the candidate nodes to decimate (assuming all have approximately equal bias), w_4 participates in the most cycles (two of the 4-cycles are shown with red and blue). Consequently, decimating w_4, as shown in Fig. 1(b), has the effect of breaking those cycles for future iterations and limiting the associated correlation in messages.

Our algorithm therefore requires knowledge of the cycle distribution of the graph. Detecting and enumerating cycles in the factor graph can be accomplished by graph theoretic methods (e.g., [11] and methods described therein) or, as we used in this work, by employing a modified BP process by incorporating additional information (meta-data) into the BP messages [12]. We remark that cycle enumeration need only be performed once (offline) prior to the DeciPolicy algorithm. From this, we create a data structure that tabulates the cycles that a variable node participates in. As the algorithm proceeds and the graph is reduced, cycles are removed from the structure, and the number of cycles per node is updated. In order to reduce complexity, in our implementation we only record cycles of length equal to the girth g and use that multiplicity to determine the node(s) to decimate. Other strategies are possible, however we have found this to be effective for all graphs considered (see Section IV).

For the cycle decimation policy, to reduce complexity, we use a more stringent filter ϕ_{bias}, which returns all undecimated nodes such that the bias is maximum (or it can be set to filter all nodes with bias larger than some threshold). Then, the node selection $\theta_{\text{cycle}}(z)$ policy returns the z nodes that participate in the most cycles from the filter. In our implementation, we only enumerate cycles of length equal to the girth for this policy.

E. The Assignment Policy χ

After identifying the variables to be decimated, the next consideration is: "What values should be assigned to the decimated variables?" The values assigned to these decimated variables are assigned using the χ function, which can be deterministic, although it still relies on the factor graph's current state. The most straightforward variable assignment mechanism following propagation involves choosing values with the highest marginal value or utility function. Indeed, [9] employs two deterministic criteria for inference: 1) Hard Decision:

$$w_i = \begin{cases} \frac{1}{2}\left(1 - \text{sign}(R_i^t)\right), & \text{if } |B_i| > 0, \\ \text{Bernoulli}\left(\frac{1}{2}\right), & \text{if } B_i = 0, \end{cases} \tag{6}$$

where i is the index of the variable that is decimated in the iteration t; and 2) Randomized Decision:

$$w_i = \begin{cases} 0, & \text{with prob } \frac{1}{2}\left(1 + \tanh(\gamma R_i^t)\right), \\ 1, & \text{with prob } \frac{1}{2}\left(1 - \tanh(\gamma R_i^t)\right). \end{cases} \tag{7}$$

In our numerical results, we will use the hard decision with bias values compared via soft decimation equations (1)-(3).

IV. Numerical Results

In this section, the results of various experiments of a C++-based implementation of Algorithm 2 are reported for different LDGM code ensembles. Codes were generated randomly following the Ising model or the optimized degree distribution from [6],

$$\lambda(x) = x^6, \tag{8}$$

$$\rho(x) = 0.275698x + 0.25537x^2 + 0.076598x^3 + 0.39233x^8. \tag{9}$$

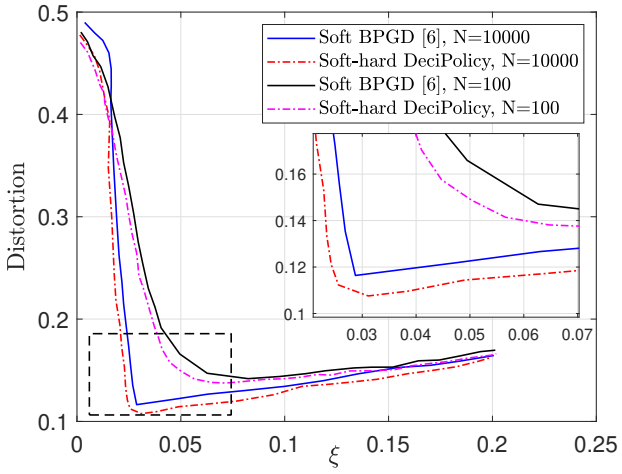

Fig. 2. BPGD distortion as a function of ξ for LDGM codes with $R = 1/2$.

The performance of different combinations of decimation policies in DeciPolicy is compared to hard BPGD [9], soft BPGD [6], and its extension soft-hard BPGD [8]. In the following examples, decimation policies are noted as a descriptive triple. For instance, DeciPolicy(bias, [cycle, 2], hard) means that decimation is triggered with the bias trigger (2 iterations with a total change less than $\epsilon = 0.01$) (τ), the cycle-detection policy with $z = 2$ variables to be decimated per iteration (ψ), and a hard value assignment (χ).

Fig. 2 compares the average distortion obtained by varying softness parameter ξ in the soft-hard BPGD using the DeciPolicy algorithm versus soft decimation [6]. The constructed LDGM code is irregular, rate $R = 1/2$, length $N = 100$ or 10000, following the optimized degree distribution (8)-(9). For this example, we set DeciPolicy(bias, [cycle, 1], soft-hard) for the soft-hard BPGD algorithm. We observe that employing cycle detection yields better distortion results for all ξ, where the corresponding optimal distortion values are $D = 0.112293$ for DeciPolicy and $D = 0.116371$ for soft BPGD.

Table I shows the performance of Algorithm 2 compared to Algorithm 1 in [8] (soft-hard BPGD) and Algorithm 1 (hard BPGD) [9] in terms of the average distortion performance. All numerical results presented are obtained by averaging over 1000 trials using the same LDGM code with length 128000 constructed from the Ising model with different check node degrees k. For all algorithms we set $I_{max} = 2000$, the decimation trigger (τ) is set with $\epsilon = 0.01$; the perform policy (ψ) for DeciPolicy is cycle detection with $z = 1$ variable decimated per iteration, whereas for [8] and [9] we use the maxbias policy with $z = 1$; and χ is hard decimation. We observe that the distortion performance of the cycle detection DeciPolicy significantly outperforms those with hard BPGD and soft-hard BPGD. For all algorithms, the performance worsens with increasing k; this behavior follows from the fact that the suboptimal BPGD threshold worsens with increasing k as a result of the cycles in the graph, while the optimal encoding threshold improves to the Shannon limit [9]. We note the gap between DeciPolicy and the others is increasing with k, possibly due to the cycle decimation being more effective

k	BPGD[9]	BPGD[8]	DeciPolicy
3	0.1536	0.1326	0.1218
4	0.1617	0.1528	0.1378
5	0.1825	0.1663	0.1504

TABLE I
AVERAGE DISTORTION RESULTS OF DECIPOLICY COMPARED TO HARD BPGD [9] AND SOFT-HARD BPGD [8].

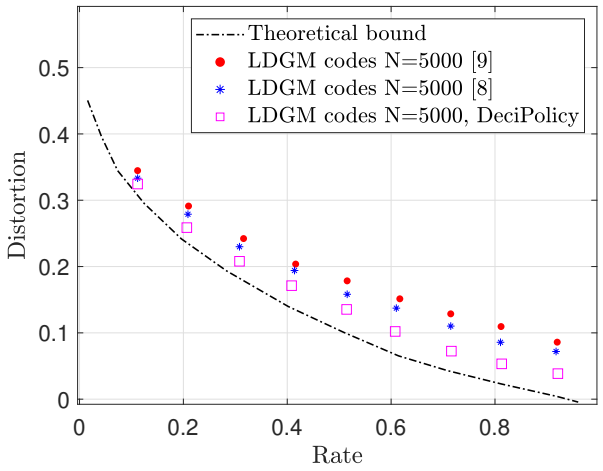

Fig. 3. Distortion performance over a range of rates for LDGM codes with BPGD and soft-hard BPGD.

with a larger number of cycles.

Fig. 3 compares the average distortion obtained from the hard and soft-hard BPGD algorithms versus soft-hard DeciPolicy as a function of R for long code lengths. The constructed codes have check node degrees $k = 4$ and length $N = 5000$. The distortion is computed for fixed R, and for 50 instances, the empirical average is taken. We compare against the RD bound $D_{sh}(R)$, shown as the lowest dotted black curve. We observe that, as expected, the distortion performance of codes over all rates has the ordering hard [9], soft-hard [8], and DeciPolicy from worst to best, with larger gains for larger R.

Finally, to reduce the number of required iterations (and therefore complexity), we compare the DeciPolicy(bias, [cycle, z], random) with different z, i.e., the $z \geq 1$ code nodes with the largest bias and most cycles are decimated each iteration. We constructed an LDGM code with a length of 2000 from the Ising model with generator node degree $k = 3$, averaging over 1000 trials. Table II shows the average distortion performance obtained of Algorithm 2 (D_2), the gain compared to Algorithm 1 (measured as $\Delta(D) = D_1 - D_2$), distortion loss for various z compared to D_1 (with $z = 1$) and finally, the complexity of the algorithm measured as the average number of updates that a code bit received over the entire process. The numerical results in Table II indicate that, by increasing the number z of decimated code nodes at each iteration (parallel decimation), the complexity decreases drastically but with a corresponding distortion loss. However, this could be desirable; for example, at $z = 10$, we observe only a 0.31% loss in distortion performance for a 87% decrease in computational complexity.

z	D_2	$\Delta(D)$	Distortion Loss	Complexity
1	0.1505	0.0210	–	991.37
2	0.1512	0.0203	0.0007	497.70
3	0.1525	0.0190	0.0020	333.21
4	0.1527	0.0188	0.0022	317.19
10	0.1536	0.0179	0.0031	122.95

TABLE II
AVERAGE DISTORTION AND COMPLEXITY RESULTS FOR DECIPOLICY WITH VARIOUS DECIMATED NODES PER ITERATION z.

V. CONCLUSIONS

We proposed the novel DeciPolicy method, which is parameterized in terms of policies to decide when to trigger decimation, which variables to decimate, and which values to assign to decimated variables. In particular, we showed that decimation should not be performed randomly, as is the convention, rather it should take the graph structure into account. By following the proposed cycle detection policy, we observe significant gains in decimation performance. The approach was verified numerically for LDGM codes constructed from both optimized irregular and semi-regular Ising models.

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grant No. CCF-2145917.

REFERENCES

[1] T. J. Goblick, "Coding for discrete information source with a distortion measure," Ph.D. dissertation, MIT, 1963.
[2] Y. Matsunaga and H. Yamamoto, "A coding theorem for lossy data compression by LDPC codes," *IEEE Trans. Inf. Theory*, vol. 49, pp. 2225–2229, 2003.
[3] M. Wainwright and E. Martinian, "Low-density graph codes that are optimal for source/channel coding and binning," *IEEE Trans. Inf. Theory*, vol. 55, no. 3, 2009.
[4] M. Wainwright, E. Maneva and E. Martinian, "Lossy Source Compression Using Low-Density Generator Matrix Codes: Analysis and Algorithms," *IEEE Trans. Inf. Theory*, vol. 56, no. 3, pp. 1351-1368, 2010.
[5] T. Filler and J. J. Fridrich, "Binary quantization using belief propagation with decimation over factor graphs of LDGM codes," CoRR, vol.abs/0710.0192, 2007.
[6] D. Castanheira and A. Gameiro, "Lossy source coding using belief propagation and soft-decimation over LDGM codes," *IEEE Int. Symp. on Personal, Indoor and Mobile Radio Comm.*, pp. 431-436, 2010.
[7] A. Golmohammadi, D. G. M. Mitchell, J. Kliewer and D. J. Costello, "Encoding of spatially coupled LDGM codes for lossy source compression," *IEEE Trans. Comm.*, vol. 66, no. 11, pp. 5691-5703, 2018.
[8] M. Alinia and D. G. M. Mitchell, "Optimizing Parameters in Soft-hard BPGD for Lossy Source Coding," Int. Symp. on Topics in Coding (ISTC), Brest, France, pp. 1-5, Sept. 2023.
[9] V. Aref, N. Macris and M. Vuffray, "Approaching the Rate-Distortion Limit With Spatial Coupling, Belief Propagation, and Decimation," *IEEE Trans. Inf. Theory*, vol. 61, no. 7, pp. 3954-3979, 2015.
[10] A. Montanari, F. Ricci-Tersenghi, and G. Semerjian, "Clusters of solutions and replica symmetry breaking in random K-satisfiability," *J. Stat. Mechanics: Theory and Experiment*, vol. 2008, no. 4, p. P04004, 2008.
[11] A. Gomez-Fonseca, R. Smarandache, and D. G. M. Mitchell, "An Efficient Strategy to Count Cycles in the Tanner Graph of Quasi-Cyclic LDPC Codes," *IEEE J. Sel. Areas in Inf. Theory*, vol. 4, pp. 499-513, 2023.
[12] M. Karimi and A. H. Banihashemi, "Message-Passing Algorithms for Counting Short Cycles in a Graph," *IEEE Trans. Comm.*, vol. 61, no. 2, pp. 485-495, Feb. 2013.
[13] Z. Sun, M. Shao, J. Chen, K. M. Wong, and X. Wu, "Achieving the Rate-Distortion Bound With Low-Density Generator Matrix Codes," *IEEE Trans. Comm.*, vol. 58, pp. 1643-1653, Jun. 2010